**Министерство науки и высшего образования Российской Федерации**
**Федеральное государственное бюджетное образовательное учреждение высшего образования**
**«Московский государственный технический университет имени Н. Э. Баумана**
**(национальный исследовательский университет)»**
**(МГТУ им. Н. Э. Баумана)**

ФАКУЛЬТЕТ «Информатика и системы управления» (ИУ)
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии» (ИУ7)

# Отчет по лабораторной работе №7 по курсу «Анализ алгоритмов»

*«Алгоритмы поиска»*

Группа: ИУ7-53Б

Студент: _____ Дьяченко А. А.
(Подпись, дата) (Фамилия И. О.)

Преподаватель: _____ Строганов Д. В.
(Подпись, дата) (Фамилия И. О.)

*Москва, 2024 г.*

## СОДЕРЖАНИЕ

## ВВЕДЕНИЕ

Задача поиска подстроки одна из достаточно распространённых в информатике. Строкой называют последовательность символов (в произвольном порядке) взятых из заданного алфавита. Так например, алфавитом могут быть цифры 0, 1 из которых можно составлять неограниченную по длине цепочку символов, например, 0110100110 или 0 [1].

Практическое значение задачи о точных совпадениях трудно преувеличить. Эта задача возникает в широком спектре приложений: текстовых редакторах, информационно-поисковых системах, электронных библиотеках, каталогах и справочниках и т. д. Алгоритмы поиска подстроки в строке также применяются при поиске заданных образцов в молекулах ДНК [2].

Целью данной лабораторной работы является исследование лучших и худших случаев работы алгоритмов поиска подстроки в строке стандартным алгоритмом и Кнута – Морриса – Пратта.

Для достижения поставленной цели необходимо решить следующие задачи:

- описать используемые алгоритмы поиска;
- выбрать средства программной реализации;
- реализовать данные алгоритмы поиска;
- проанализировать алгоритмы по количеству сравнений;
- подготовить отчет по лабораторной работе.

# 1 Аналитическая часть

## 1.1 Стандартный алгоритм

Одним из самых очевидных и одновременно неэффективных алгоритмов является алгоритм последовательно (прямого) поиска. Суть его заключается в сравнении искомой подстроки с каждым словом в основной строке. Алгоритм не является эффективным, так как максимальное количество сравнений будет равно O((n-m+1)*m), где большинство из них на самом деле лишние. Для небольших строк поиск работает довольно быстро, но если в файлах с большим количеством информации последовательность символов будет искаться очень долго [3].

## 1.2 Алгоритм Кнута — Морриса — Пратта

Метод, использующий предобработку искомой строки и создающий на ее основе префикс-функцию, содержится в алгоритме Кнута-Морриса-Пратта (КМП). Суть этой функции заключается в нахождении наибольшей подстроки, одновременно находящейся и в начале, и в конце подстроки (как префикс и как суффикс). Смысл префикс-функции заключается в том, что неверные варианты могут быть заранее отброшены, а в начале работы могут рассматриваться некоторые вспомогательные утверждения, где для произвольного слова рассматриваются все его начала, которые по совместительству являются его концами, и выбираются из них самое длинное. Метод КМП использует следующую идею: если префикс (он же суффикс) строки длиной i длиннее одного символа, то он одновременно и префикс подстроки длиной i-1. Время работы всей процедуры линейно и есть O(m), несмотря на то, что в ней присутствует вложенный цикл [4].

## 2 Конструкторская часть

В этом разделе будут представлены псевдокоды реализуемых алгоритмов.

### 2.1 КМП

Определим следующие операторы и функции:

– оператор $\leftarrow$ обозначает присваивание значение переменной;

– оператор $[i]$ обозначает получение элемента из массива с индексом $i$;

– функция $length$ возвращает длину массива, строки.

В листинге 1 рассмотрен псевдокод стандартного алгоритма.

Алгоритм 1 — Стандартный алгоритм

**Входные данные:** Строка $text$, подстрока $pattern$

**Выходные данные:** Индекс первого вхождения $pattern$ в $text$, или -1 если не найден

```
1  n ← length(text);
2  m ← length(pattern);
3  цикл i ← 1 от n − m + 1 выполнять
4  |  j ← 1;
5  |  до тех пор, пока j ≤ m and text[i + j − 1] = pattern[j]
   |   выполнять
6  |  |  j ← j + 1;
7  |  конец
8  |  если j > m тогда
9  |  |  возвратить i;
10 |  конец
11 конец
12 возвратить -1;
```

Алгоритм Кнута-Морриса-Пратта поиска подстроки в строке использует префиксную функцию. Префиксная функция, вычисляемая для искомой подстроки, содержит сведения о том, в какой мере образец совпадает сам с собой после сдвигов. Эта информация помогает избежать лишних проверок при поиске [2].

В листинге 2 рассмотрен псевдокод алгоритма расчета префиксного массива.

Алгоритм 2 — Расчет префиксного массива

**Входные данные:** Подстрока $x$

**Выходные данные:** Префиксный массив $\pi$

```
1  m ← length(x);
2  π[0] ← 0;
3  k ← 0;
4  цикл q ← 1 от m выполнять
5  |  до тех пор, пока k > 0 и x[k + 1] ≠ x[q] выполнять
6  |  |  k ← π[k];
7  |  конец
8  |  если x[k + 1] = x[q] тогда
9  |  |  k ← k + 1;
10 |  конец
11 |  π[q] ← k;
12 конец
13 возвратить π;
```

В листинге 3 рассмотрен псевдокод алгоритма Кнута – Морриса – Пратта.

Алгоритм 3 — Кнута – Морриса – Пратта

**Входные данные:** Строка $w = a_1 \dots a_n$, подстрока $x = b_1 \dots b_m$, префиксный массив $\pi$

```
1  i ← 0;
2  j ← 0;
3  n ← length(w);
4  m ← length(x);
5  до тех пор, пока i < n выполнять
6      если a_i = b_j тогда
7          i ← i + 1;
8          j ← j + 1;
9          если j = m тогда
10             возвратить i − n;
11         конец
12     конец
13     иначе
14         если j = 0 тогда
15             i ← i + 1;
16         конец
17         иначе
18             j ← π[j − 1];
19         конец
20     конец
21 конец
22 возвратить −1;
```

# 3 Технологическая часть

В данном разделе приведены требования к программному обеспечению, средства реализации и листинги кода.

## 3.1 Требования к ПО

На вход подается строка и искомая в ней подстрока. На выходе требуется получить индекс первого вхождения подстроки или отрицательное число, если такое не было найдено.

## 3.2 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран Python. Данный выбор обусловлен наличием библиотеки *time* [5] для замера времени выполнения алгоритмов.

## 3.3 Сведения о модулях программы

Программа состоит из следующих программных модулей:

1) standart.py — модуль реализации стандартного алгоритма;
2) prefix.py — модуль реализации расчета префиксного массива;
3) KMP.py — модуль реализации алгоритма Кнута – Морриса – Пратта;
4) time.py — модуль замера времени выполнения алгоритмов;
5) plot.py — модуль создания графического представления значений;
6) main.py — модуль, содержащий меню.

### 3.4 Реализация алгоритмов

В листинге 1 приведена реализация стандартного алгоритма поиска подстроки в строке.

Листинг 1 – Стандартный алгоритм

```
def standart(text, pattern):
    n = len(text)
    m = len(pattern)
    comparisons = 0

    for i in range(n - m + 1):
        j = 0
        while j < m and text[i + j] == pattern[j]:
            j += 1
            comparisons += 1
        if j == m:
            return i, comparisons

    return -1, comparisons
```

В листинге 2 приведена реализация расчета префиксного массива.

Листинг 2 – Получение префиксного массива

```
def get_prefix_array(x):
    m = len(x)
    pi = [0] * m
    k = 0

    for q in range(1, m):
        while k > 0 and x[k] != x[q]:
            k = pi[k - 1]
        if x[k] == x[q]:
            k += 1
        pi[q] = k

    return pi
```

В листинге 3 приведена реализация алгоритма Кнута – Морриса – Пратта.

Листинг 3 – Алгоритм Кнута – Морриса – Пратта

```
def kmp(text, pattern):
    n = len(text)
    m = len(pattern)
    pi = get_prefix_array(pattern)
    q = 0
    comparisons = 0

    for i in range(n):
        while q > 0 and pattern[q] != text[i]:
            q = pi[q - 1]
            comparisons += 1
        comparisons += 1
        if pattern[q] == text[i]:
            q += 1
        if q == m:
            return i - m + 1, comparisons
    return -1, comparisons
```

Алгоритм также возвращает второе значение кол-ва сравнений потому что это требуется для определения худшего случая положения искомой подстроки в тексте в исследовательской части.

# 4 Исследовательская часть

## 4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялся замерный эксперимент:

— операционная система Windows 11;

— память 16 ГБ;

— процессор 3,6 ГГц 6-ядерный процессор AMD Ryzen 5000 series 5.

Замеры проводились на ноутбуке, включенном в сеть электропитания. Во время тестирования ноутбук был нагружен только интегрированной средой разработки и непосредственно выполняемой программой.

## 4.2 Пример работы программы

На рисунке 1 представлен пример работы программы.

```
Выберите действие:
1. Ввести строку и подстроку вручную
2. Сгенерировать случайные строку и подстроку
0. Выход
Ваш выбор (1 или 2): 2
Введите длину строки: 100
Введите длину подстроки: 2
Сгенерированная строка: flkwarkbzjhzdipmfyszkzqqcievtqhgqrttxqdeaheqagiquofjyubcyfbrfnrmqpmlvjtxeqpeqfdxsptryowvgymouzjwlthy
Сгенерированная подстрока для поиска: ag
Выберите алгоритм для поиска (1 - стандартный, 2 - КМП): 1
Подстрока найдена в позиции 44
Выберите действие:
1. Ввести строку и подстроку вручную
2. Сгенерировать случайные строку и подстроку
0. Выход
Ваш выбор (1 или 2): 1
Введите строку: abacaba
Введите подстроку для поиска: ba
Выберите алгоритм для поиска (1 - стандартный, 2 - КМП): 2
Подстрока найдена в позиции 1
Выберите действие:
1. Ввести строку и подстроку вручную
2. Сгенерировать случайные строку и подстроку
0. Выход
Ваш выбор (1 или 2): 0
Выход.
```

Рисунок 1 – Пример работы программы

### 4.3 Оценка числа сравнений алгоритмов

Для обоснования, какой случай является худшим в текущей реализации алгоритмов: случай отсутствия подстроки длиной S или случай нахождения её в строке на последних S позициях, проведены соответствующие замеры.

Введём обозначения:

- СБВ — стандартный алгоритм без вхождения подстроки;
- КБВ — алгоритм КМП без вхождения подстроки;
- СВН — стандартный алгоритм с вхождением подстроки в начале текста;
- КВН — алгоритм КМП с вхождением подстроки в начале текста;
- СВК — стандартный алгоритм с вхождением подстроки в конце текста;
- КВК — алгоритм КМП с вхождением подстроки в конце текста.

В таблице 1 приведены усредненные значения кол-ва сравнений за время работы алгоритмов в 1000 испытаниях.

Таблица 1 – Число сравнений

| Длина текста | Длина подстроки | СБВ | КБВ | СВН | КВН | СВК | КВК |
|---|---|---|---|---|---|---|---|
| 256 | 8 | 12 | 268 | 8 | 8 | 18 | 265 |
| 512 | 16 | 23 | 533 | 16 | 16 | 39 | 535 |
| 1024 | 32 | 37 | 1059 | 32 | 32 | 65 | 1056 |
| 2048 | 64 | 76 | 2120 | 64 | 64 | 149 | 2129 |
| 4096 | 128 | 143 | 4242 | 128 | 128 | 288 | 4253 |
| 8192 | 256 | 303 | 8500 | 256 | 256 | 583 | 8509 |
| 16384 | 512 | 604 | 16992 | 512 | 512 | 1117 | 16970 |

Соответствующий таблице график приведён на рисунке 2

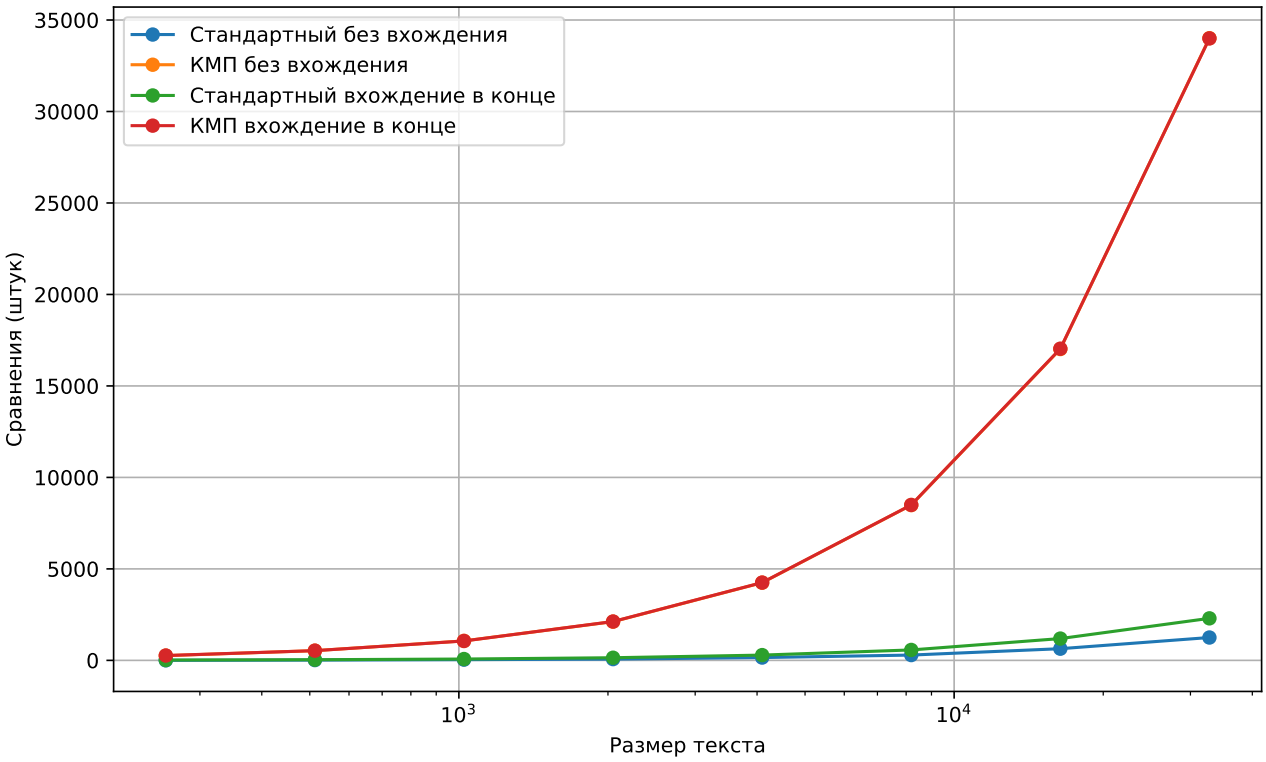


Рисунок 2 – Количество сравнений алгоритмов

Стандартный алгоритм производит больше сравнений, если подстрока присутствует в конце текста, нежели если её нет в тексте вовсе. То же самое нельзя сказать про КМП — по результатам замеров кол-во сравнений при присутствии подстроки в тексте было больше кол-ва сравнений при втором случае в $\frac{3}{8}$ случаях. Причем зависит это от того, какой текст был сгенерирован: в случае большого кол-ва совпадений префиксов подстроки с множеством подстрок текста значения в масиве префиксов больше. Следовательно, сдвиги будут больше, а кол-во сравнений — меньше.

### 4.4 Время выполнения реализованных алгоритмов

Функция perf_counter из библиотеки time [5] возвращает время в секундах — значение типа float. Является самым точным методом замера коротких промежутков времени [6].

Результаты средних значений 1000 измерений времени работы алгоритмов приведены на рисунке 3.

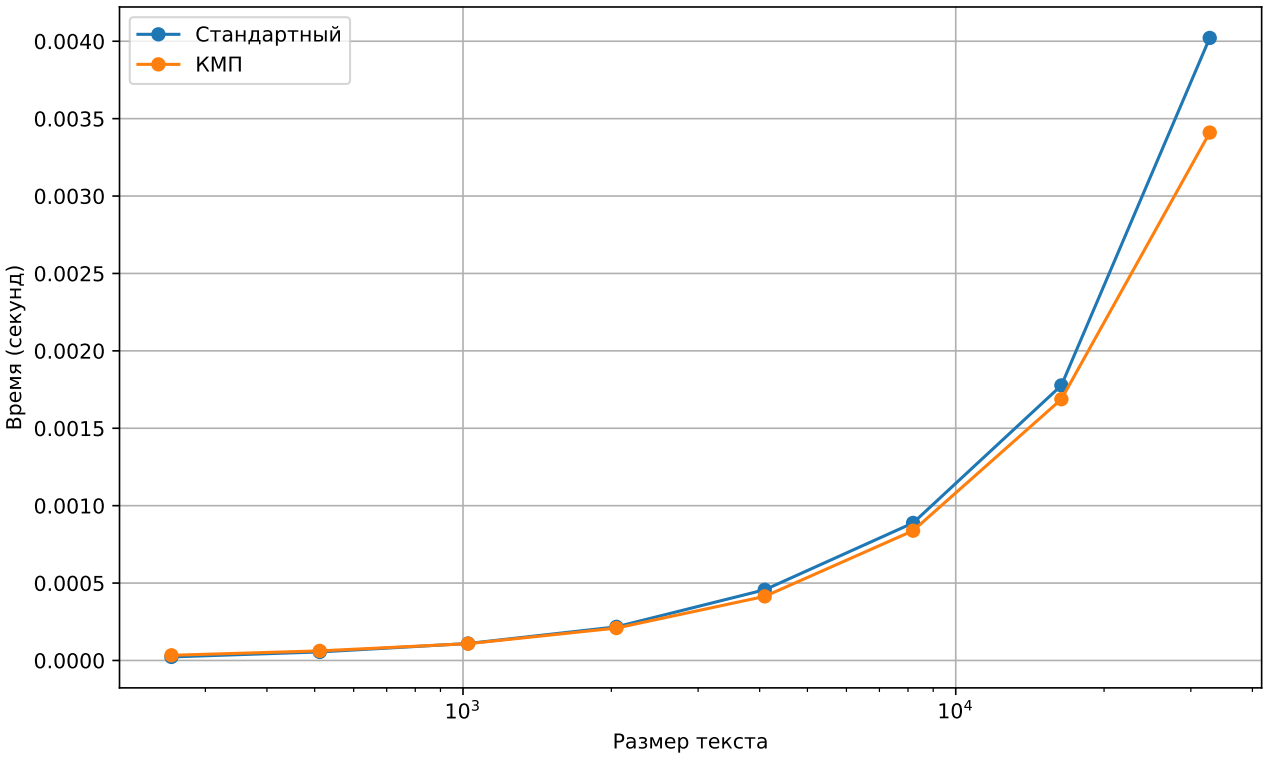


Рисунок 3 – Время выполнения алгоритмов

## Вывод

В случае отсутствия подстроки в тексте кол-во сравнений стандартного алгоритма не превышает кол-во сравнений в случае, когда строка находится в конце.

Количество сравнений в случае нахождения подстроки в конце текста и её отсутствия для алгоритма КМП практически одинаково.

При размере текста в 256 символов стандартный алгоритм быстрее алгоритма КМП на 30%. Далее, до 1000 символов разница во времени между стандартным алгоритмом и КМП меньше 11%. Для текстов длиннее 1000 символов рекомендуется использовать алгоритм КМП, т.к. по мере увеличения длины исходной строки время поиска подстроки алгоритмом КМП будет расти медленнее, чем время при поиске стандартным алгоритмом. Соответственно, чем больший размер текста — тем больший выигрышь даёт алгоритм КМП. При тексте в 32768 символов алгоритм КМП работает на 11% быстрее, чем стандартный.

## ЗАКЛЮЧЕНИЕ

Цель работы достигнута: проведено исследование лучших и худших случаев работы алгоритмов поиска подстроки в строке стандартным алгоритмом и Кнута – Морриса – Пратта.

В ходе выполнения лабораторной работы были решены все задачи:

- описаны используемые алгоритмы поиска;
- выбраны средства программной реализации;
- реализованы данные алгоритмы поиска;
- проанализированы алгоритмы по количеству сравнений;
- подготовлен отчет по лабораторной работе.

В результате анализа замеров времени выполнения был сделан вывод, что, начиная с размера текста, равного 1000, алгоритм КМП работает быстрее стандартного алгоритма. Для текста размера меньшего, чем 256 символов, рекомендуется использовать стандартный алгоритм.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. *Пелевин М.* Поиск подстроки в строке. — URL: `https://markoutte.me/students/substring`.
2. *Алексеенко А.* Информационная чувствительность алгоритма Кнута-Морриса-Пратта // Задачи системного анализа, управления и обработки информации. — 2010. — С. 5.
3. *Вирт Н.* Алгоритмы и структуры данных. Учебное пособие. — ДМК Пресс, 2010.
4. *Солдатова Г. П., Татаринов А. А., Болдырихин Н. В.* Основные алгоритмы поиска подстроки в строке // Academy. — 2018. — 5 (32). — С. 8—10.
5. *Python.* time — Time access and conversions. — URL: `https://docs.python.org/3/library/time`.
6. *Python.* perf_counter. — URL: `https://docs.python.org/3/library/time.html#time.perf_counter`.